

commodore

The Transactor

comments and bulletins
concerning your
COMMODORE PET

BULLETIN # 6
October 31, 1978

The October edition of the news bulletin begins with some very exciting news....

The complete PET business system will be unveiled at the 1978 Canadian Computer Show. On display will be the PET with a new "typewriter style" keyboard, a floppy disk, and a printer.

The Computer Show will be held in Toronto at the International Centre, 6900 Airport Road. The times and dates are:

November 28	12:00 to 6:00 P.M.
November 29	12:00 to 9:00 P.M.
November 30	12:00 to 6:00 P.M.

Snowless Version of Life:

The "Life" program listed in Transactor #5 can be further refined to eliminate the snowy effect on the screen. Mark Taylor, in the U.K., has written to us with a snowless version of Life. He also included another interesting program illustrated in example 2:

1. If you are plagued by snow on the screen during POKE operations to the screen RAM or with machine code programmes, then if in BASIC location 59409 is POKEd with 52 then that will inhibit the character generator. Any transfer operations can then be carried out with the screen totally blank. To restore normal operation the above location should be POKEd with 60. For machine code programmes then LDA 52 & STA ABSOLUTE the above location will be somewhat faster.

The attached listing is a snowless version of "Life".

```

100 READ L
110 READ A$: C=LEN(A$): IF A$="" THEN END
120 IF C<10 THEN 200
130 A=ASC(A$)-48: B=ASC(RIGHT$(A$,1))-48
140 N=B*7*(B*9)-(C=2)*(16*(A+7*(A*9)))
150 IF N<0 THEN 255 THEN 200
160 POKEL N:L=L+1: GOTO 110
200 PRINT "BYTE "L"="C"A$": " ? ? ? " : END
300 DATA 6400
310 DATA 20,30,19,20,8A,19,20,E6,19,20,00,1A,A9,34,8D,11,E8
320 DATA 20,70,19,A9,3C,8D,11,E8,A9,FF,CD,12,E8,F0,E6,4C,8B,C3,AA,68,28,4C,8B,C3
330 DATA EA,EA,EA,EA,EA,EA,EA,A2,19,BD,3A,19,95,1F,CA,D0,F8,60,00,80,00,15,00
340 DATA 80,00,1B,00,1B,D7,28,01,FE,D8,D6,29,27,00,E8,83,00,15,00,00
350 DATA EA,EA,EA,EA,EA,EA,EA,EA,EA,EA,EA,EA,EA,EA,EA,EA,EA,EA,EA,EA,EA,EA
360 DATA EA,EA,EA,EA,EA,20,A6,19,B1,26,D0,06,A9,20,91,20,D0,04,A9,51,91,20,20
370 DATA BD,19,F0,ED,20,A6,19,60,20,A6,19,B1,20,C9,51,F0,06,A9,00,91,26,F0
380 DATA 04,A9,01,91,26,20,BD,19,F0,EB,20,A6,19,60,A9,00,AA,A8,85,20,85,26,85
390 DATA 39,A5,25,85,21,A5,29,85,27,A5,36,85,3A,60,E6,26,E6,20,E6,39,E8,E4
400 DATA 33,F0,0C,E0,00,D0,0E,E6,27,E6,21,E6,3A,D0,06,A5,34,C5,21,F0,03,A9,00
410 DATA 60,A9,01,60,EA,EA,EA,EA,EA,EA,20,A6,19,B1,26,D0,06,A9,20,91,39,D0
420 DATA 04,A9,51,91,39,20,BD,19,F0,ED,20,A6,19,60,20,A6,19,20,2F,1A,B1,39,C9
430 DATA 51,F0,0C,A5,32,C9,03,D0,14,A9,01,91,26,D0,0E,A5,32,C9,03,F0,08,C9,02
440 DATA FB,04,A9,00,91,26,20,BD,19,F0,D0,20,A6,19,60,98,48,8A,48,A0,00,84,32
450 DATA A2,08,85,29,10,15,49,FF,85,37,38,A5,39,E5,37,85,22,A5,3A,85,23,B0,11
460 DATA C6,23,D0,0D,18,65,39,85,22,A5,3A,85,23,90,02,E6,23,B1,22,C9,51,D0,02
470 DATA E6,32,CA,D0,CF,68,AA,68,A8,60,*

```

2. To input data in BASIC without returning to BASIC command mode on receipt of a null string then an input statement can be simulated by a GET loop which contains additional statements to cope with DEL codes. This has the additional advantage that if there is a displayed frame on screen the frame characters will not be accepted as part of the input.

The attached listing shows the above routine. This starts at line 9000 and to use it, instead of INPUT A\$ you put GOSUB 9000:A\$=IN\$.

```

8000 REM SUBROUTINE TO SIMULATE NON-PET          STANDARD INPUT STATEMENT
8010 REM STANDARD INPUT DOES NOT BREAK          ON RECEIPT OF A NULL STRING
8020 REM "TINPUT" COULD BE ALSO BE             SIMULATED EASILY
8030 REM ZA$ IS DEFINED IN LINE 10
9000 IN$="" : PRINT " ? " :
9010 GOSUB 9070 : PRINT ZA$(ZB) : IF Z$="" THEN 9010
9020 IF Z$=CHR$(13) THEN PRINT " " : RETURN
9030 IF Z$=CHR$(20) THEN NONSGN(LEN(IN$))+1 GOTO 9010,9060
9040 PRINT Z$ : IN$=IN$+Z$ : GOTO 9010
9050 PRINT Z$ : IN$=MID$(IN$,1,LEN(IN$)-1) : GOTO 9010
9070 ZB=1+(ZB=1) : FOR ZA=1 TO 60 : GET Z$ : IF Z$<>" " THEN RETURN
9080 NEXT : RETURN

```

Programme Overlays on a PET - Supplied by Mike Stone

1. The 8K core of a PET is not usually a limitation in the home computer and hobbyist world, nor even in an educational environment where students are just creating small exercise programs. With the devices now available for attachment - the second cassette, (the printer, and floppy discs shortly) - the PET becomes a valid and genuine data processing machine, and complex string handling programs with files may well run out of space.
2. Programmers with experience on other computers know that one answer to this kind of problem is to break the program down into segments, so that only part of it is occupying memory at any time, and all or part is "overlaid" by other segments as required. When the program segments are on a disc, direct access features normally permit great flexibility, in that any required segment can be loaded; for tape only systems, the segments have to be arranged in order of need - e.g. job initialisation, main coding, and termination segments.
3. Since PET's BASIC includes a LOAD instruction to acquire dynamically a new program from tape, and (provided the new program is no longer than the one issuing the LOAD) all data areas remain available to the loaded program, the basis exists for an overlay system. However, for true overlaying, it is essential that some of the original program (e.g. control of the program flow, common subroutines, etc.) be retained throughout, whatever new segments are loaded. PET does not do this automatically; this paper tells you how to do it.

4. PET stores BASIC programs in location 1024 upwards.
Note that the pointers, and line numbets, are pairs of bytes giving low/high. The high must be multiplied by 256 and added to the low to give the actual quantity.
5. Whenever a line of code is entered from the keyboard, PET moves every statement around as necessary and re-adjusts all the chaining, so that statements are always stored in strict sequence of line number.
6. When your program contains a LOAD statement, this does NOT imply either CLR or NEW. The new program is simply loaded in at (and then executed from) location 1024, for as much space as it needs. The new program does NOT (as with some BASIC's) just replace those statements with identical line numbers; it is strictly a new program in its own right. However, any program statements at the end of the LOADING program whose space is not required by the LOADED program do remain unscathed by the LOAD. The problem is that the new program has no (forward-) chain into them, so PET knows nothing about them.
7. It follows from the above that if we code the instructions-to-be-preserved with high line numbers; and if the space needed by the newly-loaded segment does not over-write them; and if we can force the new segment to chain into the old instructions; then we have a real overlay system. So, if during the original program you can find in memory the last statement not to be preserved, you know it forward chains into the next highest line number, i.e. the first of the statements you do want preserved. Then when the overlay arrives, you need only find its very last statement and replace its forward-chain by the one you previously found, and both new and old code form a contiguous program.

8. A very simple illustration follows

Enter this program (do NOT put any spaces, except after line numbers):

```
10  A=A+1
20  GOSUB50
30  LOAD"NEWPROG"
50  PRINTA*2
55  RETURN
```

This is stored as follows ("PEEK" values):

1024)	0	
5)	11	} forward chain; $4 \times 256 = 1024 + 11 = 1035$
6)	4	
7)	10	} line number 10
8)	0	
9)	65	A
1030)	178	=
1)	65	A
2)	170	+
3)	49	1

```

4)      0
→ 5)    19      } forward chain; 4 x 256 = 1024 + 19 = 1043
6)      4      }
7)    20      } line number 20
8)      0      }
9)    141      GOSUB
1040)   53      5
1)     48      0
2)      0
→ 3)    34      } forward chain; 4 x 256 = 1024 + 34 = 1058
4)      4      }
5)    30      } line number 30
6)      0      }
7)    147      LOAD
8)     34      "
9)     78      N
1050)   69      E
1)     87      W
2)     80      P
3)     82      R
4)     79      O
5)     71      G
6)     34      "
7)      0
→ 8)    43      } forward chain; 4 x 256 = 1024 + 43 = 1067
9)      4      }
1060)   50      } line number 50
1)      0      }
2)    153      PRINT
3)     65      A
4)    172      *
5)     50      2
6)      0
7)     49      } forward chain; 4 x 256 = 1024 + 49 = 1073
8)      4      }
9)     55      } line number 55
1070)   0      }
1)    142      RETURN
2)      0
→ 3)      0

```

If we want lines 50 and 55 to be available to an overlay, the important information is the forward chain in line 30, i.e. locations 1043 and 1044.

9. To see how it works, SAVE"PROG" and leave the cassette Record and Play keys down.

Enter NEW; then the following (again, no spaces):

```
5  A=A*2
10 GOSUB50
15 STOP
```

LIST if you like, to confirm that there are no lines 50 and 55.

SAVE"NEWPROG".

Now rewind your tape, and press RUN.

PROG will be loaded, will print "2", and continue up the tape. When NEWPROG has been loaded, you will get
?UNDEF'D STATEMENT ERROR IN 10

That is because the overlay looks like this:

```
1024)  0
      5)  11      } forward chain to 1035
      6)   4      }
      7)   5      } line number 5
      8)   0      }
      9)  65      A
1030) 178      =
      1)  65      A
      2) 172      *
      3)  50      2
      4)   0
→ 5)  19      } forward chain to 1043
      6)   4      }
      7)  10      } line number 10
      8)   0      }
      9) 141      GOSUB
1040)  53      5
```

```

1)    48                0
2)    0
→ 3)    25      } forward chain to 1049
4)    4        }
5)    15      } line number 15
6)    0        }
7)   144                STOP
8)    0
→ 9)    0

```

10. The last line, 15, does not chain into the old line 50. But that line 50 is still there, in location 1058 et seq. So, do this:

```

POKE 1043,34
POKE 1044, 4

```

LIST - and behold, NEWPROG now includes lines 50 and 55!

You can RUN if you like, to prove it.

What we have done is to use what we discovered about the first program (last sentence of paragraph 9) to modify the second program.

11. How do you program all this to happen automatically? It is not at all difficult. Let us assume that the statements-to-be-preserved are at lines 5000 and upwards. So, just before that, code this (NO SPACES):

```

4997  N1=PEEK(201)    Get (low) address of line 4998
4998  N2=PEEK(202)    Get (high) address of line 4999
4999  RETURN
5000  ....  ....

```

(Locations 201, 202 always contain, during instruction execution, the address of the next instruction - strictly, the "Ø" between instructions.)

12. Now, just before your program wants to load in the overlay program, code this (spaces if you like!):

```
850 GOSUB 4997
860 N1 = N1 + 14           Low address of 4999 (the length
                           of 4998 is 14 bytes)
870 N2 = N2 * 256         Actual high address of 4999
880 If N1 < 256 THEN 900   Adjust low for page boundary
890 N1 = N1 - 256
900 BC = N1 + N2 + 1       BC is now actual machine address
                           of line 4999
910 Z1 = PEEK(BC):Z2=PEEK(BC+1) Hold the forward-chain
                           locations out of 4999
920 LOAD "NEWPROG"
```

13. As the first instructions of NEWPROG, the chain-adjusting must be done. The necessary code is very similar:

At the end of NEWPROG, as the very last statements, code (NO spaces):

```
3997      N1=PEEK(201)
3998      N2=PEEK(202)
3999      RETURN
```

And at the beginning code:

```
10      GOSUB3997
20      N1=N1+14
30      N2=N2*256
40      IF N1<256 THEN 60
50      N1=N1-256
60      BC=N1+N2+1       BC is now actual machine
                           address of 3999
70      POKE BC,Z1:POKE BC+1,Z2
```

14. It is worth just reiterating that the total size of the incoming overlay (irrespective of line numbers; just its size in bytes occupied) must be less than the total size of the instructions being overlaid.

Mike Stone

ABBREVIATING BASIC WORDS

As explained in the instruction manual, any BASIC word takes up 1 byte of memory storage space. It has been stated that the word "PRINT" can be abbreviated to "?" which saves time on entering programs. When listed, the word is expanded to its full form. Both forms take 1 byte per word.

We now have information on how to abbreviate the complete list of BASIC words. The algorithm to remember is as follows:

1. For any BASIC word, type in the first letter of the word (e.g. V for VERIFY).
2. Hold down the 'Shift' key and type in the second letter. If you are in graphics mode, this will appear as a graphic character (e.g.  for E). It is a good idea to go into lower case mode as the two letters are then easy to read. (Poke 59468, 14 → 12 for PET to graphics).

In some cases, this two-letter method gives a possibility of more than one BASIC word (e.g. READ and RESTORE). For one of the words (usually the longer) it will be necessary to type the first two letters and the shifted third. All these abbreviations are converted to full words upon the command LIST.

Below is a complete list of the words and abbreviations:

<u>BASIC</u>	<u>ABBREV</u>	<u>BASIC</u>	<u>ABBREV</u>	<u>BASIC</u>	<u>ABBREV</u>
LET	Le	DEF	De	RUN	Ru
READ	Re	RETURN	REt	CLR	Cl
PRINT	?	STOP	St	LIST	Li
PRINT#	Pr	STEP	STe	CONT	Co
DATA	Da	INPUT#	In	FRE	Fr
THEN	Th	SGN	Sg	TAB(	Ta
FOR	Fo	ABS	Ab	SPC(	Sp
NEXT	Ne	SQR	Sq	PEEK	Pe
DIM	Di	RND	Rn	POKE	Po
END	En	SIN	Si	USR	Us
GOTO	Go	ATN	At	SYS	Sy
RESTORE	REs	EXP	Ex	WAIT	Wa
GET	Ge	AND	An	LEFT\$	LEf
GOSUB	GOs	NOT	No	RIGHT\$	Ri
OPEN	Op	VAL	Va	MID\$	Mi
CLOSE	CLo	ASC	As	CHR\$	Ch
SAVE	Sa	CMD	Cm	STR\$	STr
LOAD	Lo	VERIFY	Ve		

SIMULATING A CALCULATOR ON YOUR PET

Many users have asked whether the PET can do live calculations. Although a simple sum such as $2 + 3$ can be performed thus:

```
PRINT 2 + 3 'RETURN'
```

it would be more convenient if the operation of a calculator could be simulated directly. The following program should give you an idea of how this can be achieved:

```

5 REM    GRAPHICS
10 PRINT "L"
20 PRINT " "
25 PRINT " "
30 PRINT " "
40 FOR I=1 TO 19
50 PRINT " "
60 NEXT I
1000 REM  CONTROLLER / INPUT
1010 GET A$: IF A$="" GOTO 1010
1020 A=ASC(A$)
1030 IF A>57 THEN 4000
1040 IF A<48 AND A<>46 THEN 2000
1050 IF T=1 THEN X$="" : T=0
1055 IF LEN(X$)=9 THEN D$="ERROR" : GOSUB 5115 : T=1 : GOTO 1000
1060 X$=X$+A$ : X=VAL(X$) : GOSUB 5000
1070 GOTO 1000
2000 REM  OPERATORS
2010 IF A<48 OR A=44 THEN D$="ERROR" : GOSUB 5115 : CLR : GOTO 1000
2020 IF A=48 THEN N=N+1 : B(N)=X : X=0 : Y=0 : O$(N)=O$ : O$="" : T=1 : GOSUB 5000 : GOTO 1000
2030 IF O$="*" THEN X=X*Y
2040 IF O$="/" THEN X=Y/X
2050 IF O$="+" THEN X=X+Y
2060 IF O$="-" THEN X=Y-X
2065 Y=X : O$=A$ : T=1
2070 IF A=41 THEN Y=B(N) : O$=O$(N) : N=N-1 : T=0
2080 GOSUB 5000 : GOTO 1000
4000 IF A$="S" THEN X=SIN(X)
4010 IF A$="C" THEN X=COS(X)
4020 IF A$="T" THEN X=TAN(X)
4030 IF A$="L" THEN X=LOG(X)
4040 IF A$="E" THEN X=EXP(X)
4042 IF A$="=" GOTO 2000
4043 IF A$="=" THEN CLR : T=1
4045 GOSUB 5000
4050 GOTO 1000
5000 REM  DISPLAY
5010 X$=STR$(X)
5020 D$=RIGHT$(" "+X$,11)+" "
5030 IF X<999999999 AND X>.01 GOTO 5115
5040 IF X=0 GOTO 5115
5050 IF ABS(X)>1E38 OR ABS(X)<1E-38 THEN D$="ERROR" : GOTO 5115
5100 R$=RIGHT$(" "+X$,15)
5110 D$=LEFT$(R$,11)+" "+RIGHT$(R$,3)
5115 PRINT "*****" D$
5120 RETURN
READY.

```

{ "home" 2x "CURSOR DOWN" 12x "CURSOR RIGHT" }

Although the program is by no means perfected, the framework exists for a versatile program. Lines 4000 onwards determine the functions so that when 'S' is pressed the sine of the number on display is calculated and to clear all registers '+' is pressed. The normal operation for +, -, x and ÷ is the same as a straightforward calculator, and there are multiple sets of brackets.

This idea could be used to simulate actual models - including programmable calculators, thus giving access to a wide range of ready-written low key software. We would like to hear from any user who succeeds in doing this.

BITS AND PIECES

Some more hints and tips to help you write efficient programs:

When writing REMark statements, graphics and lower case can be included if they are put inside inverted comma's. This enables separating lines such as:

```
10  REM "_____"
```

```
*      *      *      *      *      *      *      *      *      *      *
```

When using subscripted variables such as A(4) the operating system automatically reserves 10 elements without having to declare a dimension with DIM. If, however, you are using a very long program and are using less than 10 elements per variable - say 4 - it will save space to declare the dimension's length. For example:

```
10  DIM A(4), C$(3)
```

```
*      *      *      *      *      *      *      *      *      *      *
```

To display a number (N) to D decimal places, use the following routine:

```
10  M = INT(N*10^D+.5)/10^D
```

```
20  PRINT M
```

```
*      *      *      *      *      *      *      *      *      *      *
```

For an intriguing display of graphics, try running this one line program entitled "BURROW"

```
1  A$="↑↓→←":PRINTMID$(A$,RND(.5)*4+1,1)"*←";:FORT=1TO30:  
NEXT:PRINT"Ⓜ←";:GOTO1
```

```
*      *      *      *      *      *      *      *      *      *      *
```

NEW PET MANUAL:

We are currently working on an extensive PET users manual. The manual will eventually be available for sale through our dealers. We do plan, however, offering the manual in chapters through future issues of the "Transactor".